Polling and Events

L. Pivetta

http://www.tango.controls.org

Polling

- Polling a device means **periodically executing** some of the methods and storing the results in a **buffer**
- All the necessary parameters are stored in the TANGO database
  - the configuration is kept between device restarts
- The aim is:
  - Decouple the equipment (hardware) from the client(s)
  - Speed up response time for slow devices
  - Maintain some history of device command(s) execution/attribute(s) reading
  - Be the data source for the TANGO event subsystem (more on this later)
- In addition polling allows to continuously monitor the “health” of a device
- It is possible to configure polling in order to poll:
  - Any command **without** input parameters
  - Any attribute

Polling

- Each TANGO device has its own polling buffer
  - The polling buffer has configurable depth, just limited by available memory
  - In fact each Attribute/Command has its own polling buffer, but
  - The polling buffer depth is the same for all Attributes/Commands

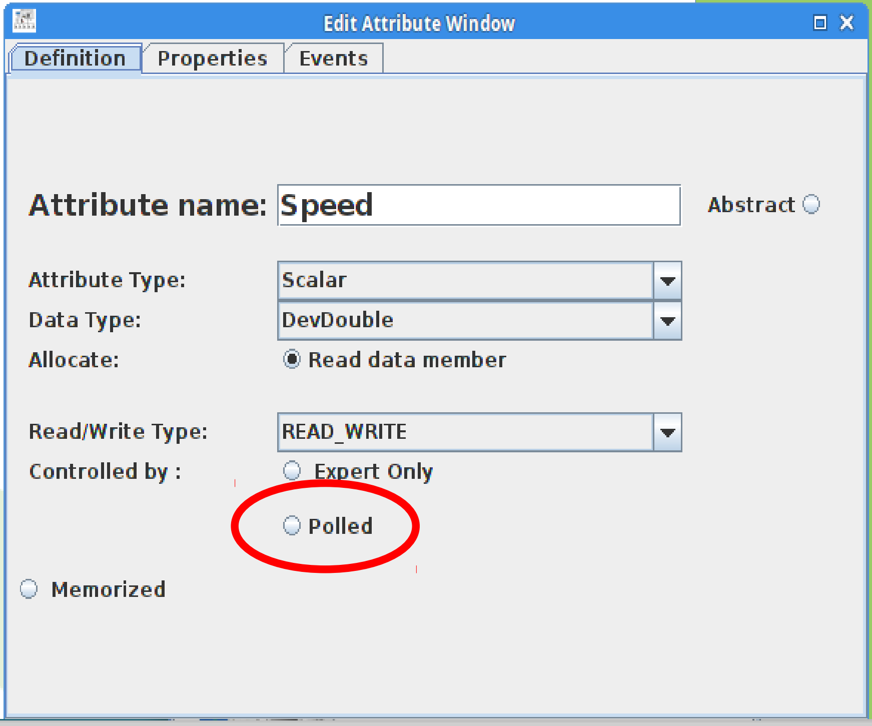
- A client is able to read data from:
  - The real equipment (DEVICE)
  - The last record in the polling buffer (CACHE)
  - The polling buffer with fall-back to the real equipment (CACHE_DEVICE)

- The complete buffer history is also available to the client
  - Large buffers mean a sort of “automatic” shared memory mechanism available

Polling

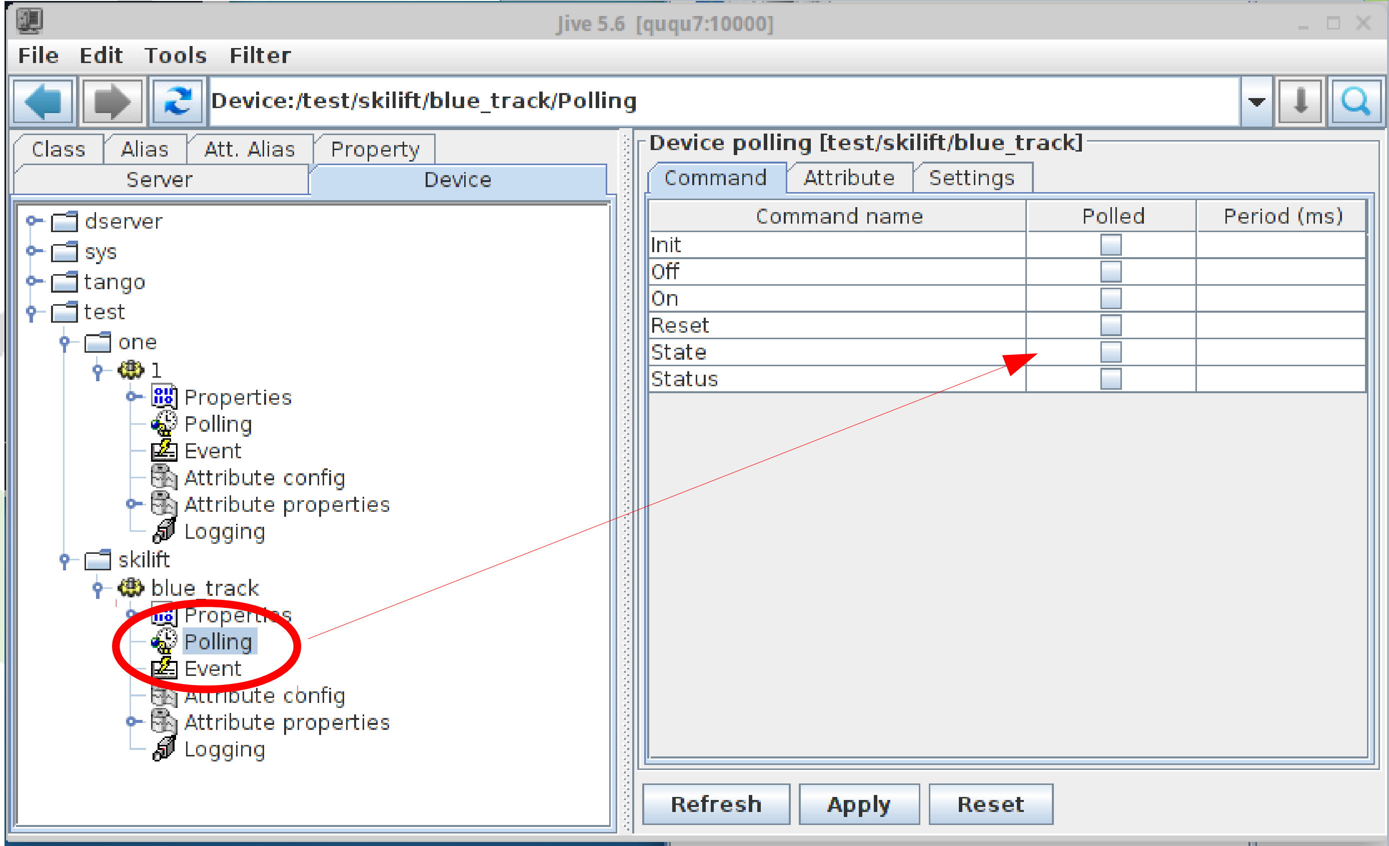
How to setup polling?

- During Class modeling with POGO, using the check-buttons
- At runtime, configuring the TANGO database with Jive
- Programmatically, using, for instance Python with the client API
- Programmatically, in the device itself



Attribute name: Speed   Abstract ○

Attribute Type: Scalar
Data Type: DevDouble
Allocate: ● Read data member
Read/Write Type: READ_WRITE
Controlled by : ○ Expert Only
○ Polled
○ Memorized



Device:/test/skilift/blue_track/Polling

Device polling [test/skilift/blue_track]

Command name	Polled	Period (ms)
Init	☐	
Off	☐	
On	☐	
Reset	☐	
State	☐	
Status	☐	

Polling

- Polling configuration/management is done sending commands to the administration device, provided by the TANGO core for every device

- Seven commands available:
  - AddObjPolling – add a new attr/cmd to the list of polled obj, specify polling period
  - RemObjPolling – remove cmd/attr from the list of polled obj
  - UpdObjPollingPeriod – change cmd/attr polling period
  - StartPolling – start polling for the whole device
  - StopPolling – stop polling for the whole device
  - PolledDevice – query for polling
  - DevPollStatus – detailed polling status

Polling

- Polling period is specified per device attribute/command

- If the polling period is the same for all polled attrs/cmds, then:
  - Equipment access frequency≅ 1 / ((polling period) x (nr. of polled cmds/attrs))
    - Polling period = 1s
    - Nr. of polled cmds/attrs = 10
    - Equipment access every 100ms roughly

(True if the device implementation sticks to the basic model having each method accessing the equipment, e.g. no "read attribute hardware", "always executed hook", "timestamp check" or "custom scheduling" tricks…)

- Always keep in mind real equipment access time/limitations when setting up polling

- Keep some spare CPU time for the device to to other jobs (...clients?)
  - At least 50%

- It's useless (and inconvenient) setting up 1s polling period for 10 attributes on a TANGO device talking to a serial line equipment that takes 200ms round-trip (and even 100ms round trip)

- TANGO uses **one polling buffer depth** per device for all attrs/cmds

Polling thread(s) pool

- Starting with TANGO release 7, a Tango device **server process** may have several polling threads managed as a pool

- Useful in case of:
  - several devices within the same device server process,
  - accessing different hardware equipments
  - with very different access times

    (or when one equipment is not responding thus generating long timeout and de-synchronizing the polling thread)

- The polling thread pool can be managed
  - with a GUI, available in the administration Tools
  - acting on the TANGO administration device

Polling thread(s) pool

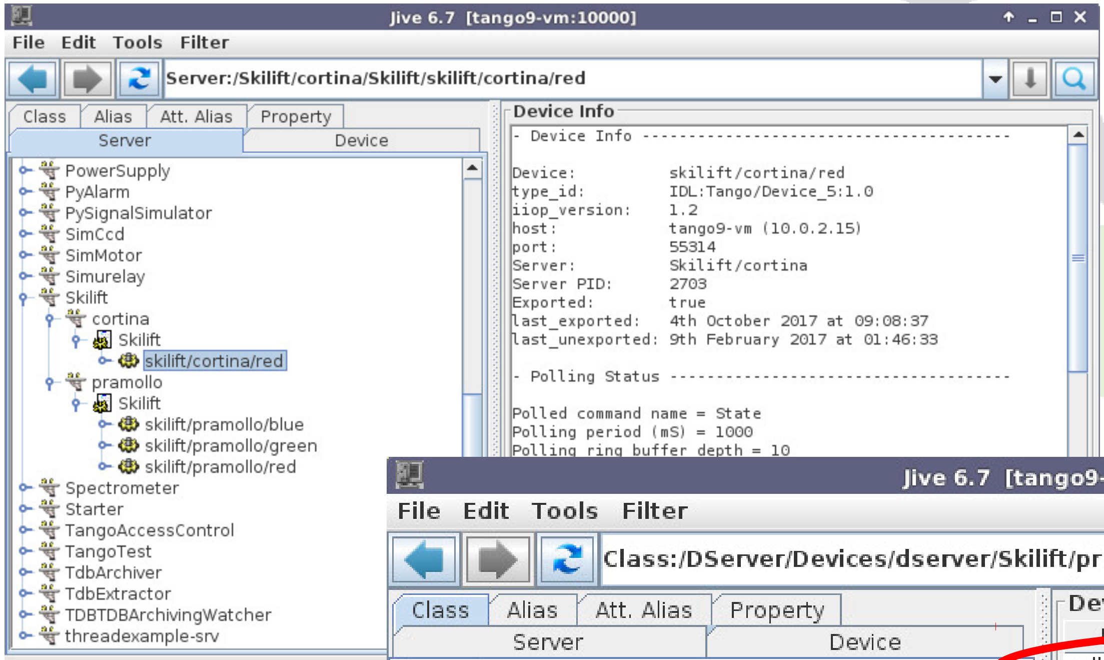
- Two specific properties:
  - *polling_threads_pool_size*: max number of threads in the pool
  - *polling_threads_pool_conf*: device/thread map

- N.B. the **granularity** of the polling thread pool tuning is the **device**
  - You cannot use two threads to poll two different attributes belonging to the same device

- When you ask for an object, e.g. cmd/attr, polling:
  - If there are already polled objects for the device, the new one is added to the thread (managing this device)
  - If there is no thread for the device
    - If the number of threads is less than *polling_threads_pool_size,* then a new thread is created
    - Otherwise the object is added to the thread with the smallest number of polled objects

Remember:
- **All** devices in the same **device server** process share the polling threads pool
- The default is **one** polling thread for all devices in the device server process

Polling thread(s) pool

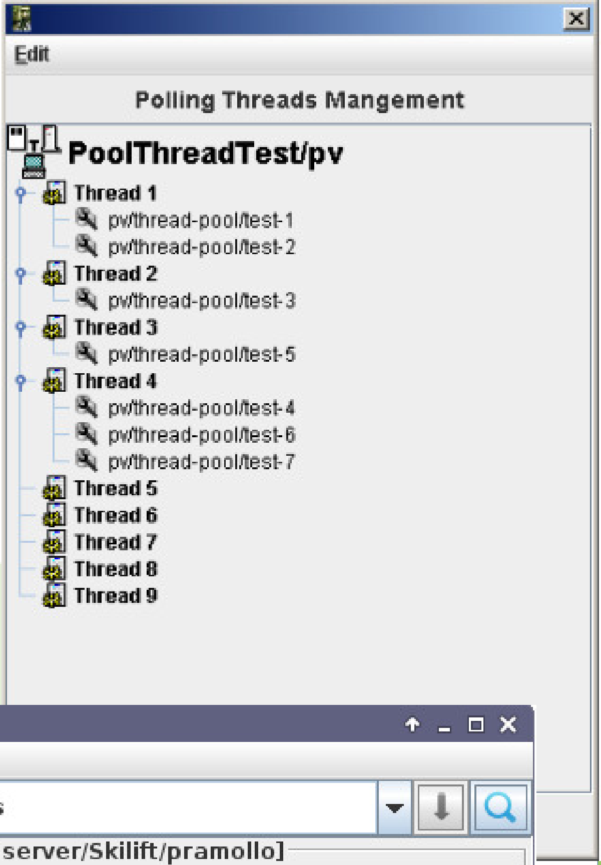


Jive 6.7 [tango9-vm:10000]

Server:/Skilift/cortina/Skilift/skilift/cortina/red

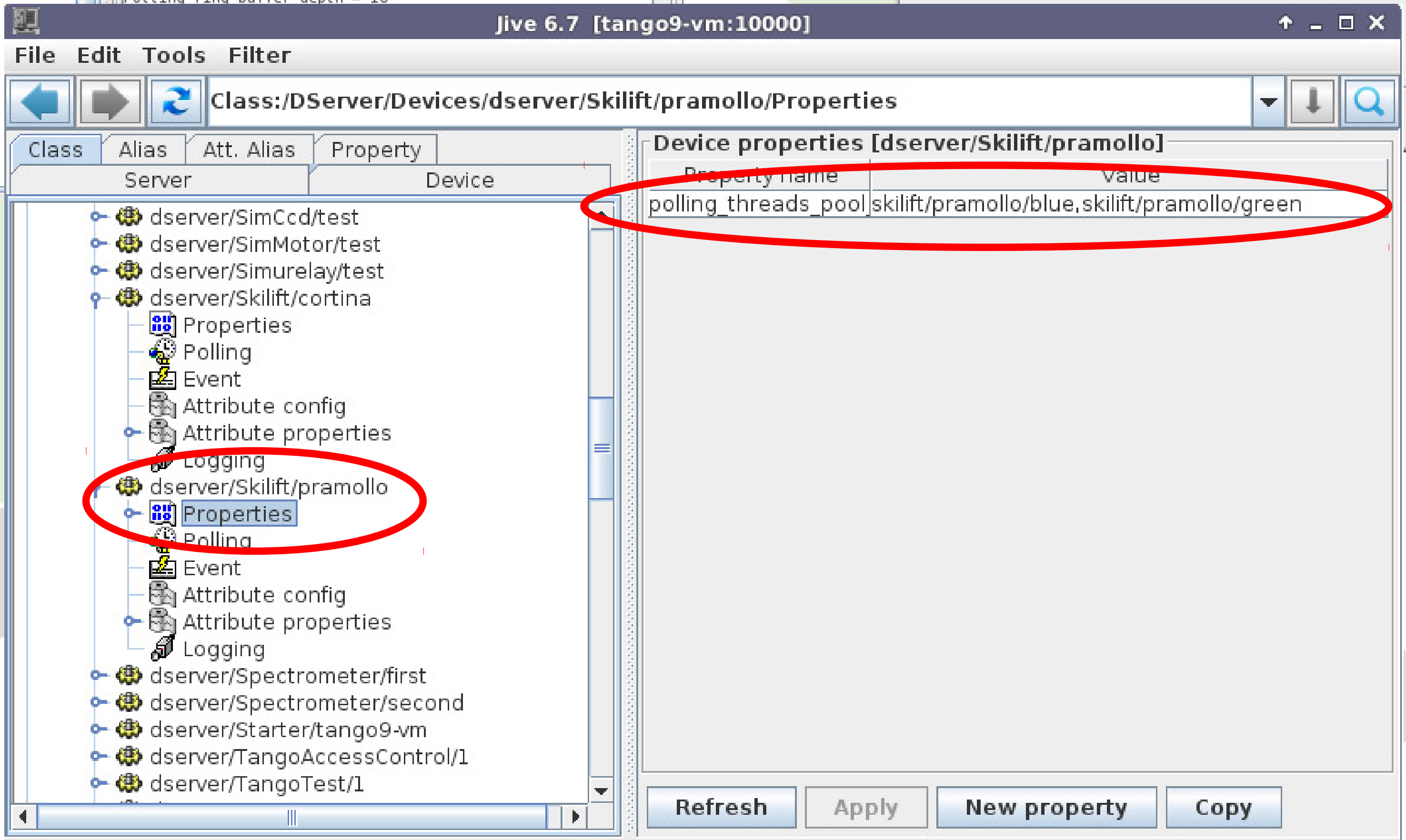
```
- Device Info ------------------------------------

Device:           skilift/cortina/red
type_id:          IDL:Tango/Device_5:1.0
iiop_version:     1.2
host:             tango9-vm (10.0.2.15)
port:             55314
Server:           Skilift/cortina
Server PID:       2703
Exported:         true
last_exported:    4th October 2017 at 09:08:37
last_unexported:  9th February 2017 at 01:46:33

- Polling Status ----------------------------------

Polled command name = State
Polling period (mS) = 1000
Polling ring buffer depth = 10
```

- PowerSupply
- PyAlarm
- PySignalSimulator
- SimCcd
- SimMotor
- Simurelay
- Skilift
  - cortina
    - Skilift
      - skilift/cortina/red
  - pramollo
    - Skilift
      - skilift/pramollo/blue
      - skilift/pramollo/green
      - skilift/pramollo/red
- Spectrometer
- Starter
- TangoAccessControl
- TangoTest
- TdbArchiver
- TdbExtractor
- TDBTDBArchivingWatcher
- threadexample-srv



Polling Threads Mangement

- PoolThreadTest/pv
  - Thread 1
    - pv/thread-pool/test-1
    - pv/thread-pool/test-2
  - Thread 2
    - pv/thread-pool/test-3
  - Thread 3
    - pv/thread-pool/test-5
  - Thread 4
    - pv/thread-pool/test-4
    - pv/thread-pool/test-6
    - pv/thread-pool/test-7
  - Thread 5
  - Thread 6
  - Thread 7
  - Thread 8
  - Thread 9



Jive 6.7 [tango9-vm:10000]

Class:/DServer/Devices/dserver/Skilift/pramollo/Properties

Device properties [dserver/Skilift/pramollo]

Property name	Value
polling_threads_pool	skilift/pramollo/blue,skilift/pramollo/green

- dserver/SimCcd/test
- dserver/SimMotor/test
- dserver/Simurelay/test
- dserver/Skilift/cortina
  - Properties
  - Polling
  - Event
  - Attribute config
  - Attribute properties
  - Logging
- dserver/Skilift/pramollo
  - Properties
  - Polling
  - Event
  - Attribute config
  - Attribute properties
  - Logging
- dserver/Spectrometer/first
- dserver/Spectrometer/second
- dserver/Starter/tango9-vm
- dserver/TangoAccessControl/1
- dserver/TangoTest/1

http://www.tango-controls.org

Polling thread(s) pool

- Up to **TANGO 8** the polling thread uses *read_attribute()* on each polled attribute
  - → TANGO monitor locking on each call

```
/FOR/ each attribute to be read
    /CALL/ always_executed_hook()
    /CALL/ read_attr_hardware()
    /CALL/ is_<xxx>_allowed()
    /IF/ previous call returns true
        /CALL/ read_<xxx>()
    /ENDIF/
/ENDFOR/
```

- In **TANGO 9** will use *read_attributes()* **if attributes have the same polling period**
  - TANGO monitor locking

```
/CALL/ always_executed_hook()          ← just once
/CALL/ read_attr_hardware()            ← just once
/FOR/ each attribute to be read
    /CALL/ is_<xxx>_allowed()
    /IF/ previous call returns true
        /CALL/ read_<xxx>()
    /ENDIF/
/ENDFOR/
```

- The polling thread is allowed to be “late”
- You can revert to TANGO 8 behavior
  (polling_before_9=true propertyin the admin device)

Polling

- The TANGO framework **needs** the polling to work well
- Each client reading a device triggers independent execution of device methods, possibly down to the hardware access
- …and, starting clients is very easy with TANGO
- ...and, basically, you have no control over (people) running clients
- **So, you need polling**
- Which attributes/commands to poll?
  - All that need fast client access
  - All that have slow response time (slow hardware, slow method[1]...) but...
  - **Just one client reading one non-polled attribute (...yes, unfortunately exactly that slow one...) can break your device**

[1] – anyhow, each method execution time need to be taken into account

Polling

- Then, what to do? (open discussion)
  - Poll everything (unused attributes very slowly)
  - Cache unused attributes by hand (e.g. hand-made scheduling in the *always_executed_hook)*
  - Use additional custom thread(s)
  - Otherwise, be careful with clients

- Still, remember:
  - The **client can force** reading the device selecting the DEVICE source
  - And the CACHE_DEVICE possible fall back

Events

- Implement the publish/subscribe pattern; **based on ZeroMQ since Tango 8** (no more notification service)
- Available on **attributes**
- The client registers her interest **once** in an event (value)
- The server informs the client every time an event has occurred
- **Default based on device server polling**: needs configuration but does not require changes in the device server code
- Additionally the event generation can be managed by the developer: **events pushed by code**
- Client callback executed when an event is received
- Eight (six up to TANGO 8) types of events available:
  - **Change**: absolute change, relative change
  - **Periodic**: period
  - **Archive**: absolute change, relative change, period
  - **Attribute configuration**: no parameters
  - **Data ready**: managed by the developer
  - **User**: managed by the developer
  - **Device interface change**: managed by the kernel (TANGO 9)
  - **Pipe**: managed by the developer (TANGO 9)

When are events pushed?

- **Change event**
  - at event subscription
  - a change is detected in attribute data
  - a change is detected in attribute size (spectrum/image)
  - the attribute quality factor changes
  - exception in the polling thread
  - by set_quality() and set_value_date_quality() methods (*)
- **Periodic event**
  - at event subscription
  - on a periodic basis
- **Archive event**
  - a mix of periodic and change
- **Attribute configuration event**
  - at event subscription
  - the attribute configuration is modified
- **User defined event**
  - when the user decides
- **Device interface change** (Tango 9)
  - when the device interface changes, e.g.:
    - when dynamic commands/attributes added/removed
    - at device restart or reconnection
- **Pipe** (Tango 9)
  - when the programmer decides (code *DeviceImpl::push_pipe_event()*)

Events configuration and behavior

Change event configuration

- Checked at the polling period
- Two thresholds: **rel_change** and **abs_change**
  Up to 2 values per threshold (positive and negative delta)
  If both set, rel_change is checked first
  If none set → **no change event**

Archive event configuration

- Checked at the polling period
- Two thresholds: **archive_rel_change**, **archive_abs_change**
  Up to 2 values per threshold (positive and negative delta)
  If both set, rel_change is checked first
  If none set → **no archive event on change**
- **archive_period [ms]**
  Default *None* → **no periodic archive event**

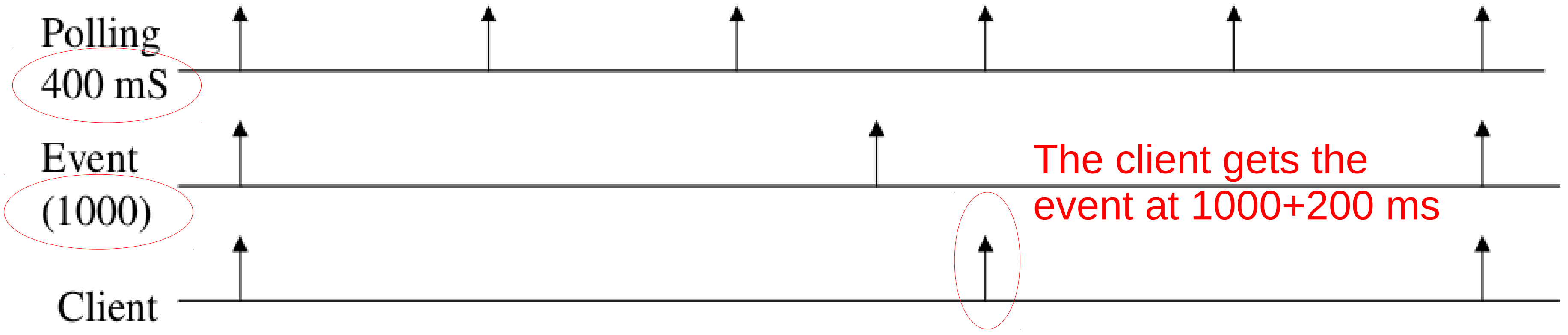
Periodic event configuration

- period [ms]

Periodic event configuration

event_period [ms]
- default value is 1000 ms
- cannot be faster than the polling period

Advice: whenever **event_period != polling period** (or multiple of)
- the event system does **not** change the attribute polling period
- the event is sent when polling occurs



Push events in the code to squeeze the best performance from the event system
Drawback: you need to write some code...

Event configuration

- Event period shorter than polling period makes no sense

- Be aware that **polling period is unique** for the Attribute/Command
  - If you setup an Attribute polling period according to the archiving requirements, lets say somehow “slow” because it's enough to check every 10s, it will affect also the *change_event* behavior
  - This means a “normal” client, e.g. a GUI, will refresh at 10s rate!!!

- Events filtering (managed by the notification service) is no more available since TANGO 8

- Push events by code if you cannot compromise

- (*) the *Attribute::set_quality()* and *Attribute::set_value_date_quality()* methods trigger **change** event emission and allows to overcome the delay possibly introduced by the polling thread

Events pushed in the code

- *change*, *data_ready* and *archive* events can be pushed by code
- The programmer decides when to push events
- The server has to **declare** events pushed by code:

- *Attr::set_change_event(bool implemented, bool detect = true);*
- *Attr::set_archive_event(bool implemented, bool detect = true);*
- *Attr::set_data_ready_event(bool implemented);*

- Where *implemented* = *true* means events are pushed by code and *detect* whether to check for event thresholds before pushing (optional)

- Than the server can call the following methods to push specific events:

- *DeviceImpl::push_change_event(string attr_name, ….);*
- *DeviceImpl::push_archive_event(string attr_name, ….);*
- *DeviceImpl::push_data_ready_event(string attr_name, ….);*

- A special call is available to push non standard events (see TANGO user manual page 178-179)

Events heartbeat

Heartbeat

- To check that the device server is alive

  Every 10 seconds a special heartbeat event is sent to all clients on the event channel

- To inform the server that no more clients are interested in events

  A re-subscription command is sent by the client every 200 seconds.

  The device server stops sending events as soon as the last subscription command is older than 600 seconds

- A dedicated client thread (keep-alive thread) wakes up every 10 seconds to check the server's 10 seconds heartbeat and to send the subscription command periodically